**Electrical Calendar Project Proposal**

By Henry Tang (heweit2), Gezhi Zou (gezhiz2), Joey Zhou (weijian4), Zijun Gao (zijung3)

**Pitch**

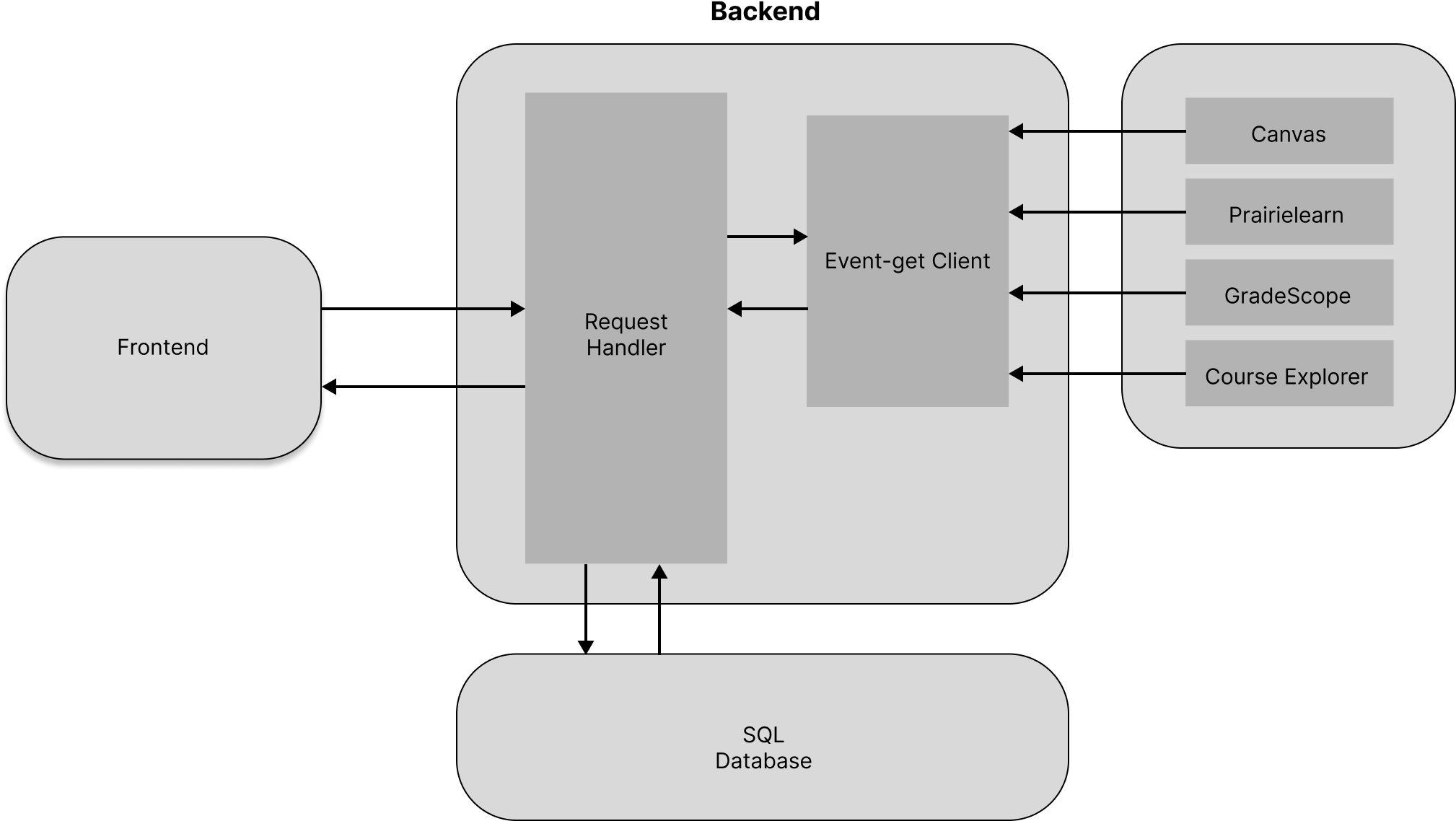
Electrical Calendar makes time management an easier job for college students by seamlessly connecting multiple online education platforms, offering a comprehensive interface to quickly view weekly agendas, and ensuring students can effortlessly organize and prioritize their academic responsibilities.

**Functionality**

1. Users can synchronize deadlines of assignments from various platforms such as Canvas, Moodle, Gradescope, etc.
2. Users can synchronize their course schedule
3. Users can set reminders of events
4. Users can create independent events
5. Users can invite others and share information about events
6. Users can see a universal calendar including all events from deadlines to independent events.

**Components**

* Backend: We will write the backend using Python with the Django framework. The backend will handle HTTP requests following the OAuth specification, ensuring secure access control. The real-time interaction required for synchronizing schedules and sending reminders will be achieved through WebSockets, enabling efficient, bidirectional communication between the frontend and backend without the overhead of constant polling.
* The backend has following responsibility:
  + Store the data: We would like to use SQL like database to store data
    - Universal events and events created by users: SQL database
    - Calendar created by users
    - Users' authentication data about accounts: Django Auth
    - Data can be accessed by different users
  + Send reminders to users
    - We can send emails or Pop-up-Warning
  + Share information between users
    - Users can invite other to joint events
    - Users can send short messages
  + Access and get events
    - Get time about deadlines and course schedule from different website
* Frontend: We will write frontend using JavaScript and React framework. In the frontend, we will show our UI design to the user such as a visualized calendar. We will use React to create a week/ month/ day view of the calendar. Also, we will display the event created by users and different deadlines from the course website. We intend to utilize the unittest framework for validating the functionality of individual Python functions. Additionally, we will develop test cases aimed at evaluating the application's overall performance.
* The architecture is designed to be modular, allowing each component to be developed, tested, and deployed independently. It also supports scalability, as each layer can be scaled independently to handle increased load. The separation of concerns makes the system more maintainable and reduces complexity for developers working on individual components.
  + The frontend would make requests to the backend to retrieve or modify the calendar data as the user interacts with the application, so that users can view their calendar, create events, edit details, and so forth.
  + Request handler is responsible for receiving information from the front-end and assigning it to different modules for execution. When the module has finished executing, it will send the processed information back to the front-end and store information into the database. Based on this, we can achieve information interactions between users.
  + Event-get Client gets the information needed by the user by calling api of different websites, fed to Request handler, then Request handler feeds back to the front-end and writes it into the database.

****

**Weekly Planning**

| **Week # (Date)** | **Tasks** |
| --- | --- |
| Week 1 (2/19-2/23) | * Install all libraries for the backend. * Setup frontend environment * Find all api used in Event-get Client Module |
| Week 2 (2/26-3/1) | * Create basic data structure of calendar and SQL Database * Create the data structure of calendar in the backend |
| Week 3 (3/4-3/8) | * Create basic data structure of events * Combine both calendar and events |
| Week 4 (3/18-3/22) | Writing different functions to grab informations from websites:   * Canvas * Prairielearn * Gradescope * Course Explorer |
| Week 5 (3/25-3/29) | * Using React to design a UI view of the calendar |
| Week 6 (4/1-4/5) | * Call backend method from frontend to store data in SQL * Call backend method to get proper data from the database |
| Week 7 (4/8-4/12) | Apply the feature for user interactions:   * Message transfer * Event invitation for other users |
| Week 8 (4/15-4/19) | * Backend testing |
| Week 9 (4/22-4/26) | * Frontend testing * Decorate the frontend page |
| Week 10 (4/29-5/3) | * Clean up all small features * Extra buffer time for any risks |

**Potential Risks**

**1. Risk: Limited API Access**

The challenge of potentially not being able to access APIs from applications like Canvas, Gradescope, and PrairieLearn, which hinders the direct retrieval of task deadlines, will be countered by allocating 1-2 weeks to seek and integrate any available APIs; should this prove unfeasible, we're prepared to extend the initial development phase by up to 3 weeks to create and test a user interface for manual data input.

**2. Risk: Email Notification System**

The uncertainty involved in the creation of an automated email notification system for timely class and deadline reminders will be approached by dedicating a research period to understanding similar systems such as Course Explorer's Notify Me feature, with a planned 2-week buffer following the system's anticipated completion for troubleshooting and potential mentor consultation, and a willingness to redirect efforts towards an in-app notification system if necessary.

**3. Risk: App Development Learning Curve**

The risk posed by the team's inexperience in app development, potentially underestimating the complexity of the project, will be mitigated by establishing a comprehensive learning plan leveraging existing resources and open-source frameworks, with a flexible project timeline that includes specific milestones for learning phases, and readiness to redistribute tasks or seek further educational support if progress is delayed.

**Teamwork**

To address the lack of a standardized development environment, our team will implement a Docker-based workflow. This will provide a consistent set of development tools across different systems, thereby avoiding the common issue of discrepancies in package management among our team members’ various operating systems. Additionally, Docker allows team members to choose their preferred IDEs or text editors, accommodating personal development preferences while maintaining a unified backend environment.

To streamline our collaborative efforts, we have scheduled weekly team meetings. During these sessions, each member will present their progress from the previous week and outline their objectives for the upcoming week. This structured exchange will help synchronize our efforts, enhance accountability, and facilitate effective collaboration.

For project execution, we will adopt a dual-subteam structure, pairing team members to work on specific project components, promoting peer oversight, and enhancing communication:

Front-end Subteam: Gezhi and Weijian will focus on crafting the application’s user interface and interaction design. Their collaboration will ensure a cohesive user experience and allow for shared creative input.

Back-end Subteam: Hewei and Zijun will concentrate on establishing the database architecture and integrating API connections. This division ensures that our application’s backbone is robust and reliable.

This preliminary subteam framework is flexible; we will assess and realign the team configuration as needed based on real-time progress to ensure a balanced workload and steady project advancement. This approach allows us to adapt to unforeseen challenges while leveraging individual strengths within the team.